AcroTEX.Net

The acromemory Package

D. P. Story

Prepared: June 26, 2020

www.acrotex.net
Version 2.0, 2020/06/23

Table of Contents

1. Introduction

AcroMemory (acromemory) is a memory game in which you find the matching tiles. There are two versions—available as options of this package—for your enjoyment, `acromemory1` and `acromemory2` (the default). Only one option is allowed to be taken for any document, only one memory puzzle per document is supported. (In theory, you can have multiple instances of the same memory puzzle, but I don't really see a need for that.)

- `acromemory1`: Here you have a single game board, a rectangular region divided by rows and columns. The total number of tiles *must* be even, each tile *must* have a matching twin. The game begins with all the tiles hidden. The user clicks a tile, then another. If the tiles do not match, they become hidden again (you did remember the position of those tiles, didn't you?); otherwise, they remain visible and are now read-only. The game is complete when the user, with a lot of time on his/her hands, matches all tiles. There is a running tabulation kept on the number of tries. There is also a button which resets the game and randomizes the tiles.

- `acromemory2`: For this game you have two identical rectangular images subdivided into tiles (or slices), which are arrayed in rows and columns. The tiles for one of the two images is randomly re-arranged. The object of the game is to find all the matching tiles by choosing a tile from one image and a tile from the other image. As in the first case, if the selected tiles do not match, they are hidden after a short interval of time (you did remember the position of those tiles, didn't you?); otherwise, they remain visible and are now read-only. The game is over when all tiles are matched; when this occurs, end-of-game special effects occur that will dazzle the senses. There is an option to view a small image to help you locate the matching tiles on the non-randomized; useful if the image is complex.

The demo files. These are `acromemory1.tex` and `acromemory2.tex`. These files show how to lay out the various elements of this package.

Supported workflows. This new version of acromemory is a complete re-write of the old version. All the common workflows are supported: pdflatex, lualatex, xelatex, and dvips -> distiller.

2. Package Options

There are a few options of this package:

- `acromemory1` and `acromemory2`: As described earlier. The `acromemory2` option is the default.

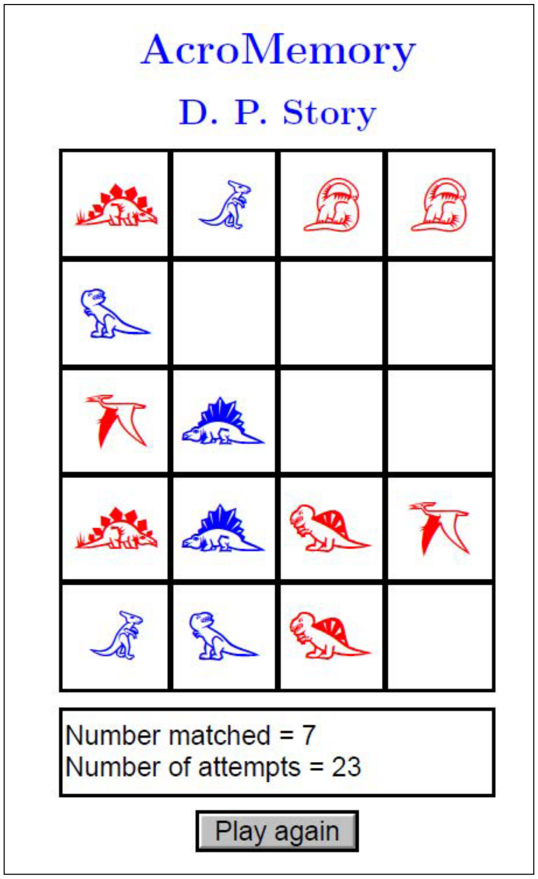
- `draft`: The `draft` option is passed to the graphicx package. Useful when setting up the layout of your document and when composing the document. Works for pdflatex and lualatex PDF creators.

- `includehelp`: When building a file with the acromemory2 option, you can also include a help image, a small picture of the game board to help the user to match the randomized tiles with the ones on the non-randomized game board. Useful if the image is very complex. The demo file `acromemory2.tex` contains the necessary code for producing the help feature, the commands only create the help feature if the `includehelp` option is taken.

3. Commands of the Package

We describe the commands of this package as well as methodology for creating tiled graphics for use by this package.

3.1. Commands for the acromemory1 option



Number matched = 7
Number of attempts = 23

Figure 1: Partially worked acromemory1 puzzle

The preamble. For the `acromemory1` option, the preamble begins with some variation of the following:

Using the web package

```
  \documentclass{article}
① \usepackage[%
①   web=designv,useacrobat
① ]{aeb_pro}
② %\usepackage[designv]{web}
  \usepackage[acromemory1]{acromemory}
```

Without using the web package

```
\documentclass{article}
\usepackage[acromemory1]{acromemory}
```

The lines ① are optional, the advantage of using the `aeb_pro` package is that it enables the example files to be compiled *using all workflows*: pdflatex, lualatex, xelatex, and dvips -> distiller. The alternative is to use ②, that is, the web package with a design option, in this case `designv`. On the right is the appropriate declarations without using web. All sample files use the web package.

Embedding the tiles. The next step is to embed the tiles using the `embedding` environment of the icon-appr package:[1]

```
\begin{embedding}
%\isPackage
\amEmbedTiles{dinos}{10}{dinos/mydinos}
\end{embedding}
```

The `\isPackage` and `\amEmbedTiles` commands are used for embedding the tiles.

```
\isPackage
\amEmbedTiles{⟨name⟩}{⟨n-tiles⟩}{⟨path⟩}
```

were ⟨*name*⟩ is the name of the tile set; ⟨*n-tiles*⟩ is the number of *distinct tiles*, and ⟨*path*⟩ is the path to the base name of the graphics. Note that for puzzles of type `acromemory1`, the tiles are doubled to make a total of 2 × ⟨*n-tiles*⟩ tiles on the board.

Comments on the tiled files required. There are two ways the tiles are delivered:

1. As a single package file, in which case the acromemory package looks for the file ⟨*path*⟩`_package.pdf`. The tiles are the pages of the package document. Signal to acromemory that the tiles are packaged by including `\isPackage` prior to the `\embedTiles` command. For example, from the `acromemory1.tex` demo file,

   ```
   \begin{embedding}
   \isPackage
   \amEmbedTiles{dinos}{10}{dinos/mydinos}
   \end{embedding}
   ```

   acromemory looks for the file `mydinos_package.pdf` in the `dinos` folder.

   Note. This feature is only available for the pdflatex, lualatex, and dvips -> distiller workflows.

---

[1] https://ctan.org/pkg/icon-appr

2. If the files are not packaged, acromemory looks for the files ⟨*path*⟩`_01.pdf`, ⟨*path*⟩`_02.pdf`, …, ⟨*path*⟩`_10.pdf`, …. This method is available to all workflows, including xelatex.

In addition to the tiles themselves, packaged or not, acromemory needs one or two more files. We illustrate using the dinos example. For the dvips -> distiller workflow, acromemory looks for `mydinos.eps`; for all other workflows, it looks for `mydinos.pdf`. These files are used to measure the dimensions of a typical tile. The two basic files `mydinos.eps` and `mydinos.pdf` are just one of the tiles, perhaps it is just `mydinos_01` as an EPS or PDF file.[2]

In the body of the document. After the tiles have been embedded (in the preamble), we are ready to insert the tiles into the document with `\insertTiles`:

```
\insertTiles{dinos}{2in}{5}{4}
```

The syntax for `\insertTiles` is,

```
\insertTiles{⟨name⟩}{⟨width⟩}{⟨n-rows⟩}{⟨n-cols⟩}
```

where ⟨*name*⟩ matches the ⟨*name*⟩ to an earlier embedding; ⟨*width*⟩ is the width of your puzzle board; ⟨*n-rows*⟩ is the number of rows; and ⟨*n-cols*⟩ is the number of columns. Of course, ⟨*n-rows*⟩ × ⟨*n-cols*⟩ = 2 × ⟨*n-tiles*⟩.

Other controls. There a two other controls to mention.

```
\messageBox[⟨eforms-opts⟩]{⟨wd⟩}{⟨ht⟩}
\playItAgain[⟨eforms-opts⟩]{⟨wd⟩}{⟨ht⟩}
```

`\messageBox` is a text field that displays various running scores; while `\playItAgain` is a push button for starting the matching game.

The results of all this effort are seen in Figure 1, or by compiling, playing, and enjoying the file `acromemory1.tex`.

Creating the tiles. This contents of these paragraphs can be skipped over on first reading. The sample file `acromemory1.tex` uses the 'Mini Pics Lil Dinos' font set; you need not have the font set itself, the glyphs are contained in the mydinos support files.

In this paragraph, we discuss the methodology for creating your own tile files. Two support files were developed to create the required files:

- `myDinos_package.tex` is used to create `myDinos_package.tex`, required for the pdflatex, lualatex, and dvips -> distiller workflows. This file should be studied to create your own package file with some other font set.
- `myDinos_files.tex` is used to create the individual files,

---

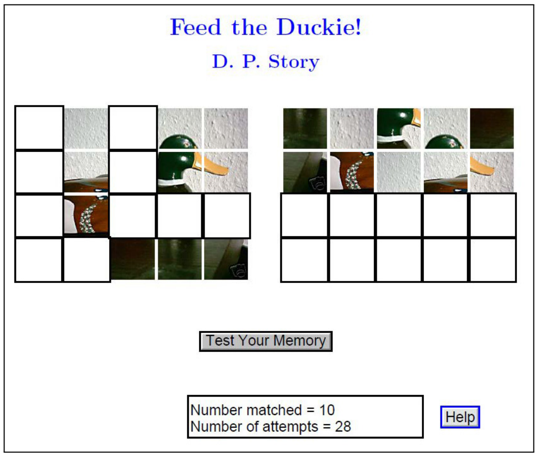
[2] If you never use the dvips -> distiller workflow, EPS file is not needed.

`myDinos_01.pdf`, `myDinos_02.pdf`, …, `myDinos_10.pdf`

as well as the two additional files `myDinos.pdf` and `myDinos.eps`. Again, this file should be studied and modified to create these individual files for your own font.

You need not use a font set to create the tile files, you can use a series of small graphics instead. We leave this as an exercise.

3.2. Commands for the acromemory2 option



Feed the Duckie!

D. P. Story

Number matched = 10
Number of attempts = 28

Figure 2: Partially worked `acromemory2` puzzle

This option produces a two board matching game. Figure 2 is a snapshot of the demo file `acromemory2.tex`.

Creating the required (tiled) graphics. For this option you don't have to have a fancy font, any (interesting) picture will do. The first step is to decide how many rows and columns you want and then tile the image appropriately. To tile the graphic, use the `tile-graphic` package.[3] In the `examples/duckie` folder, you will find `duckie-tg.tex`, which was used to produce all the files needed to create the demo file `acromemory2.pdf`. Compile `duckie-tg.tex` using your preferred workflow to produce all required files.

Let's walk through the components of `acromemory2.tex`.

The preamble. For the `acromemory2` option, the preamble begins with some variation of the following:

---

[3] `https://ctan.org/pkg/tile-graphic`

Using the web package

```
  \documentclass{article}
① \usepackage[%
①   web=designv,useacrobat
① ]{aeb_pro}
② %\usepackage[designv]{web}
  \usepackage[includehelp]{acromemory}
```

Without using the web package

```
\documentclass{article}
\usepackage[includehelp]{acromemory}
```

The lines ① are optional, the advantage of using the `aeb_pro` package is that it enables the example files to be compiled *using all workflows*: pdflatex, lualatex, xelatex, and dvips -> distiller. The alternative is to use ②, that is, the `web` package with a design option, in this case `designv`. On the right is the appropriate declarations without using `web`. All sample files use the `web` package. The use of the `includehelp` option is optional.

Embedding the tiles. The next step is to embed the tiles using the `embedding` environment of the `icon-appr` package:[4]

```
\begin{embedding}
%\isPackage
\amEmbedTiles{duck}{20}{duckie/duckie}
\end{embedding}
```

The `\isPackage` and `\amEmbedTiles` commands are used for embedding the tiles.

```
\isPackage
\amEmbedTiles{⟨name⟩}{⟨n-tiles⟩}{⟨path⟩}
```

were ⟨*name*⟩ is the name of the tile set; ⟨*n-tiles*⟩ is the number of *tiles*, and ⟨*path*⟩ is the path to the base name of the graphics.

In addition to the tiles themselves, packaged or not, `acromemory` needs one or two more files. We illustrate using the duckie example. For the dvips -> distiller workflow, `acromemory` looks for `duckie.eps`; for all other workflows, it looks for `dickie.pdf`. These files are used to measure the dimensions of the image.[5]

In the body of the document. After the tiles have been embedded (in the preamble), we are ready to insert the tiles into the document. Keep in mind, there are two puzzle boards: the left one which is the non-randomize tiled picture, and the one on the right which is the randomized tiled picture. The two commands `\insertTilesL` and `\insertTilesR` insert the left and right puzzle boards, respectively. For example,

```
\insertTilesL{duck}{2in}{4}{5}\qquad
\insertTilesR{duck}{2in}{4}{5}
```

The syntax for `\insertTilesL` and `\insertTilesR` is,

---

[4] https://ctan.org/pkg/icon-appr

[5] If you never use the dvips ->distiller workflow, EPS file is not needed.

```
\insertTilesL{⟨name⟩}{⟨width⟩}{⟨n-rows⟩}{⟨n-cols⟩}
\insertTilesR{⟨name⟩}{⟨width⟩}{⟨n-rows⟩}{⟨n-cols⟩}
```

where ⟨*name*⟩ matches the ⟨*name*⟩ to an earlier embedding; ⟨*width*⟩ is the width of your puzzle board; ⟨*n-rows*⟩ is the number of rows; and ⟨*n-cols*⟩ is the number of columns. Of course, ⟨*n-rows*⟩ × ⟨*n-cols*⟩ = ⟨*n-tiles*⟩.

Other controls. There a two other controls to mention.

```
\tryItAgain[⟨eforms-opts⟩]{⟨wd⟩}{⟨ht⟩}
\messageBox[⟨eforms-opts⟩]{⟨wd⟩}{⟨ht⟩}
\helpImage[⟨eforms-opts⟩]{⟨wd⟩}{⟨ht⟩}
\rolloverHelpButton[⟨eforms-opts⟩]{⟨wd⟩}{⟨ht⟩}
```

`\messageBox` is a text field that displays various running scores; while `\tryItAgain` is a push button for starting the matching game. The other two commands `\helpImage` and `\rolloverHelpButton` are used when the `includehelp` option is taken. These two commands do nothing if `includehelp` is not specified.

The results of all this effort are seen in Figure 2, or by compiling, playing, and enjoying the file `acromemory2.tex`.

3.3. Commands common to both options

When the user begins to work on the puzzle(s) without first pressing the 'Play again' or the 'Test your Memory' button, an alert box with the one of the following messages appear, depending on whether `acromemory1` or `acromemory2` option is in force:

```
\newcommand{\initFirstiMsg}{"Press the 'Play again'
  button to initialize the puzzle"}
\newcommand{\initFirstiiMsg}{"Press the 'Test Your Memory'
  button to initialize the puzzle"}
```

'Play again' and 'Test your Memory' are the default captions to the `\playItAgain` and `\tryItAgain` controls. Should you change the caption of, for example, `\playItAgain`, you would then redefine `\initFirstiMsg`:

```
\playItAgain[\CA{Play!}]{}{12pt}
\renewcommand{\initFirstiMsg}{"Press the 'Play!' button to
  initialize the puzzle"}
```

Or, perhaps redefine the captions and messages for some other language.

4. Final comments

Use any of the demo files as a template to create your own memory game. Don't forget (use your memory?) that the web package has options to apply a background color or a graphic—this will jazz up your memory game.

I do hope you find this game package fun, and that you will be creative in its use. Perhaps you can apply the techniques of this package to create your own game package, there are many possibilities.

Now, I simply must get back to my retirement! DPS